CIRCULO
Windows / C++ 1.0

AUTORES:

HUMBERTO ANDRES CARRILLO CALVET
ANTONIO CARRILLO LEDESMA
OSCAR RAFAEL GARCIA REGIS
MIGUEL ANGEL MENDOZA REYES
FERNANDO ALBERTO ONGAY LARIOS

México D.F. Marzo 2003

CIRCULO

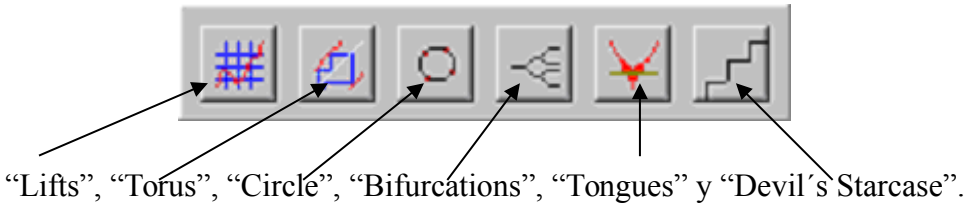
Descripción de las capacidades del sistema:

Círculo es un sistema de software que sirve para analizar sistemas dinámicos discretos en la circunferencia. Estos sistemas están gobernados por las iteraciones de una función de la circunferencia en la circunferencia. Muchos fenómenos de interés en Ciencia e Ingeniería pueden ser modelados por medio de sistemas dinámicos en la circunferencia. En particular este es el caso de muchos problemas interesantes de la teoría de oscilaciones no lineales. El sistema Círculo ha sido desarrollado usando la metodología orientada a objetos.

El sistema permite analizar la dinámica de estos sistemas y analizar los distintos comportamientos que se pueden producir cuando se varian los valores de las condiciones iniciales y los parámetros. El sistema permite: un análisis visual basado en el cálculo y despliegue de las graficas de los levantamientos de las funciones de la circunferencia y de las graficas de las funciones en el toro plano; el despliegue de órbitas, cálculo del número de rotación, cálculo de los índices de los atractores periódicos (sincronizaciones) y cálculo de los exponentes de Lyapounov entre otras funciones que se relacionan en la descripción siguiente.

La ventana principal del programa tiene la siguente apariencia:

Como puede obserbarse, Círculo se compone de seis escenarios gráficos, cada uno de los cuales sirve para analizar la dinámica que determinan los parámetros en turno del sistema, cada uno desde un punto de vista diferente. Estos escenarios son:



"Lifts", "Torus", "Circle", "Bifurcations", "Tongues" y "Devil´s Starcase".

A continuación se hace una breve descripción de cada uno:

 **Lifts Scenery**. En este escenario se traza la gráfica de los levantamientos asociados a la función de la circunferencia generada por los osciladores forzados periódicamente (función de tiempos de disparo).

 **Torus Scenery**. Muestra la gráfica de la función de fases de disparo en un rectángulo, tal que al ser identificadas de forma conveniente sus orillas es topológicamente equivalente con un toro. Por esta razón este espacio es llamado "toro plano". Puesto que la función de fases de disparo es una función de la circunferencia en la circunferencia, el toro es el espacio natural donde esta debe de ser representada.

 **Circle Scenery**. Muestra las iteraciones en la circunferencia de la función de fases de disparo. Esto permite identificar en qué fases del forzamiento periódico ocurren los disparos de la neurona.

 **Bifurcations Scenery**. En este escenario se puede ver la evolución de los atractores de las órbitas o secuencias de disparos que generan los osciladores neuronales al ser modificado gradualmente alguno de los parámetros que gobiernan a los sistemas. Cuando ocurre algún cambio cualitativo en la dinámica del sistema esta es llamada una bifurcación. En la parte principal de la ventana de este escenario se traza el diagrama de bifurcaciones y en la parte superior se traza una gráfica que representa el exponente de Lyapunov en cada punto de los parámetros. Los exponentes de Lypaunov típicamente sirven para distinguir entre dinámicas periódicas y caóticas del sistema.

 **Tongues Escenery**. En este escenario se calculan y se muestran diagramas en dos dimensiones de las regiones del espacio de parámetros donde el número de rotación es constante o donde el sistema muestra comportamientos sincronizados del mismo tipo. A estas regiones se les conoce como lenguas de Arnold.

 **Devil's Staircase Scenery**. Traza una gráfica donde se calcula el número de rotación en función de alguno de los parámetros del sistema. Típicamente se ve como una función creciente escalonada, donde cada escalón representa un conjunto de parámetros donde el número de rotación es constante.

Diez Primeras Páginas

```
////////////////////////////////////////////////////////////////////////////////////////
// CIRCULO WINDOWS C++ 1.0                                                            //
//                                                                                    //
// Análisis y Diseño:                                                                 //
//                                                                                    //
// Humberto Andres Carrillo Calvet                                                    //
// Antonio Carrillo Ledesma                                                           //
// Oscar García Regis                                                                 //
// Miguel Angel Mendoza Reyes                                                         //
// Fernando Alberto Ongay Larios                                                      //
//                                                                                    //
// Programación:                                                                      //
//                                                                                    //
// Humberto Andres Carrillo Calvet                                                    //
// Antonio Carrillo Ledesma                                                           //
//                                                                                    //
////////////////////////////////////////////////////////////////////////////////////////

#include <vcl.h>
#pragma hdrstop
USERES("Circulo.res");
USEFORM("VentanaPrincipal.cpp", FormaPrincipal);
USEUNIT("FuncionDisparo.cpp");
USEUNIT("DefiniciónEcuación.cpp");
USEUNIT("Sincronizaciones.cpp");
USEUNIT("NumeroRotacion.cpp");
USEUNIT("ExponenteLyapunov.cpp");
USEFORM("VentanaBifurcaciones.cpp", FormaBifurcaciones);
USEFORM("VentanaCirculo.cpp", FormaCirculo);
USEFORM("VentanaToro.cpp", FormaToro);
USEFORM("VentanaLevantamientos.cpp", FormaLevantamientos);
USEFORM("Acecade.cpp", VAcercaDe);
USEFORM("ParamBifurcaciones.cpp", VCap_Bifurcaciones);
USEFORM("ParamToro.cpp", VCap_Toro);
USEFORM("ParamLevantamientos.cpp", VCap_Levantamientos);
USEFORM("ParamCirculo.cpp", VCap_Circunferencia);
USEFORM("ParamGlobales.cpp", VCap_General);
USEUNIT("..\Libreria\V_pixel.cpp");
USEFORM("..\Libreria\FORMAS\Ayuda.cpp", AyudaForm);
USEFORM("..\Libreria\FORMAS\Editor.cpp", EditorForm);
USEUNIT("..\Libreria\ctrl_l_a.cpp");
USEUNIT("..\Libreria\Cadenas.cpp");
USEUNIT("..\Libreria\Fechas.cpp");
USEUNIT("..\Libreria\fechora.cpp");
USEUNIT("..\Libreria\Man_arch.cpp");
USEUNIT("..\Libreria\Tiempo.cpp");
USEFORM("..\Libreria\FORMAS\ACX-WebBrowser.cpp", FormaWebBrowzer);
USEFORM("..\Libreria\FORMAS\EnviarCorreo.cpp", FormaEnviarCorreo);
USEFORM("..\Libreria\FORMAS\Lmensaje.cpp", LMensajeForm);
USEFORM("..\Libreria\FORMAS\VEditor.cpp", VentanaCaptura);
USEFORM("ParamLenguas.cpp", VCap_Lenguas);
USEFORM("VentanaLenguas.cpp", FormaLenguas);
USEFORM("VentanaEscalera.cpp", FormaEscalera);
USEFORM("ParamEscalera.cpp", VCap_Escalera);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                 Application->Initialize();
                 Application->Title = "";
                 Application->CreateForm(__classid(TFormaPrincipal), &FormaPrincipal);
                 Application->Run();
        }
        catch (Exception &exception)
        {
                 Application->ShowException(&exception);
        }
        return 0;
}
//---------------------------------------------------------------------------



////////////////////////////////////////////////////////////////////////////////////////
// CIRCULO WINDOWS C++ 1.0                                                            //
//                                                                                    //
// Análisis y Diseño:                                                                 //
//                                                                                    //
// Humberto Andres Carrillo Calvet                                                    //
// Antonio Carrillo Ledesma                                                           //
```

```
// Oscar García Regis                                                                    //
// Miguel Angel Mendoza Reyes                                                            //
// Fernando Alberto Ongay Larios                                                         //
//                                                                                       //
// Programación:                                                                         //
//                                                                                       //
// Humberto Andres Carrillo Calvet                                                       //
// Antonio Carrillo Ledesma                                                              //
//                                                                                       //
///////////////////////////////////////////////////////////////////////////////////////////


#ifndef VentanaPrincipalH
#define VentanaPrincipalH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Menus.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
#include "DefinicionEcuacion.hpp"
#include "ParamGlobales.h"
#include <Dialogs.hpp>
#include <ExtDlgs.hpp>

//---------------------------------------------------------------------------
class TFormaPrincipal : public TForm
{
   __published:  // IDE-managed Components
        TStatusBar *BarraDeEstadoPrincipal;
        TMainMenu *Menu;
        TMenuItem *MenuArchivo;
        TMenuItem *MenuArchivoTerminarPrograma;
        TMenuItem *MenuConfigurar;
        TMenuItem *MenuAyuda;
        TPanel *PanelEscenarios;
        TPanel *PanelIconosAccion;
        TSpeedButton *IconoEscenarioEscalera;
        TSpeedButton *IconoEscenarioLevantamientos;
        TSpeedButton *IconoEscenarioToro;
        TSpeedButton *IconoEscenarioCirculo;
        TSpeedButton *IconoEscenarioBifurcaciones;
        TMenuItem *MenuAyudaAcercade;
        TMenuItem *MenuAyudaGeneral;
        TMenuItem *MenuAyudaBifurcaciones;
        TMenuItem *MenuAyudaCirculo;
        TMenuItem *MenuAyudaToro;
        TMenuItem *MenuAyudaCurvasIntegrales;
        TMenuItem *MenuAyudaLevantamientos;
        TMenuItem *MenuConfigurarNumRotacion;
        TMenuItem *MenuConfigurarSincronizacion;
        TMenuItem *MenuConfigurarExpLyapunov;
        TSpeedButton *IconoAccionNumRotacion;
        TSpeedButton *IconoAccionSincronizacion;
        TSpeedButton *IconoAccionExpLyapunov;
        TSpeedButton *IconoAccionConfigurarEscenario;
        TSpeedButton *IconoAccionCalcular;
        TSpeedButton *IconoAccionDetenerCalculo;
        TSpeedButton *IconoAccionLimpiarVentana;
        TSpeedButton *IconoEscenarioTongues;
        TMenuItem *Separador1;
        TMenuItem *MenuArchivoPrintsetup;
        TPrinterSetupDialog *PrinterSetupDialog;
        TMenuItem *MenuScenery;
        TMenuItem *MenuSceneryLifts;
        TMenuItem *MenuSceneryTorous;
        TMenuItem *MenuSceneryCircle;
        TMenuItem *MenuSceneryBifurcation;
        TMenuItem *MenuSceneryTongle;
        TMenuItem *MenuCalculate;
        TMenuItem *MenuCalculateRotationnumber;
        TMenuItem *MenuCalculateSynchronization;
        TMenuItem *MenuCalculateLyapunovExponent;
        TMenuItem *MenuCalculateStartcalculations;
        TMenuItem *MenuCalculateStopcalculations;
        TMenuItem *MenuWindowErasescenery;
        TMenuItem *Separador2;
```

```
        TMenuItem *Separador3;
        TMenuItem *MenuConfigurarScenery;
        TMenuItem *MenuWindow;
        TMenuItem *Separador4;
        TMenuItem *MenuAyudaLenguas;
        TMenuItem *Separador6;
        TMenuItem *MenuArchivoEditFile;
        TMenuItem *Separador8;
        TMenuItem *MenuArchivoPrintFile;
        TOpenPictureDialog *OpenPictureDialog;
        TMenuItem *Separador7;
        TMenuItem *MenuVentanaConformalZoom;
        TMenuItem *N1;
        TMenuItem *LabDinmicanoLineal1;
        TMenuItem *DevilsStaircase1;
        void __fastcall MenuArchivoTerminarProgramaClick(TObject *Sender);
        void __fastcall IconoEscenarioBifurcacionesClick(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall IconoEscenarioCirculoClick(TObject *Sender);
        void __fastcall IconoEscenarioToroClick(TObject *Sender);
        void __fastcall IconoEscenarioEscaleraClick(
          TObject *Sender);
        void __fastcall IconoEscenarioLevantamientosClick(TObject *Sender);
        void __fastcall MenuAyudaAcercadeClick(TObject *Sender);
        void __fastcall MenuAyudaGeneralClick(TObject *Sender);
        void __fastcall MenuAyudaBifurcacionesClick(TObject *Sender);
        void __fastcall MenuAyudaCirculoClick(TObject *Sender);
        void __fastcall MenuAyudaToroClick(TObject *Sender);
        void __fastcall MenuAyudaCurvasIntegralesClick(TObject *Sender);
        void __fastcall MenuAyudaLevantamientosClick(TObject *Sender);
        void __fastcall MenuConfigurarNumRotacionClick(TObject *Sender);
        void __fastcall MenuConfigurarSincronizacionClick(TObject *Sender);
        void __fastcall MenuConfigurarExpLyapunovClick(TObject *Sender);
        void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
        void __fastcall IconoAccionLimpiarVentanaClick(TObject *Sender);
        void __fastcall IconoAccionNumRotacionClick(TObject *Sender);
        void __fastcall IconoAccionSincronizacionClick(TObject *Sender);
        void __fastcall IconoAccionExpLyapunovClick(TObject *Sender);
        void __fastcall IconoAccionConfigurarEscenarioClick(
          TObject *Sender);
        void __fastcall IconoAccionCalcularClick(TObject *Sender);
        void __fastcall IconoAccionDetenerCalculoClick(TObject *Sender);
        void __fastcall IconoEscenarioTonguesClick(TObject *Sender);
        void __fastcall MenuArchivoPrintsetupClick(TObject *Sender);
        void __fastcall MenuAyudaLenguasClick(TObject *Sender);
        void __fastcall MenuArchivoEditFileClick(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall MenuArchivoPrintFileClick(TObject *Sender);
        void __fastcall MenuVentanaConformalZoomClick(TObject *Sender);
        void __fastcall LabDinmicanoLineal1Click(TObject *Sender);
    private:      // User declarations
        // Objeto definición de la ecuación
        DefinicionEcuacion def_ecu;
        char               TituloAplicacion[250];

        // Puntero a la ventana de configuracion
        TVCap_General     *configura;
        char               xcad[200];

        // Controla la visualizacion de Hint en la barra de estado
        void __fastcall OnHint(TObject *Sender);
        void            PasarValoresConfiguracion(void);
        void            RetornarValoresConfiguracion(void);

    public:               // User declarations
        __fastcall TFormaPrincipal(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TFormaPrincipal *FormaPrincipal;
//---------------------------------------------------------------------------
#endif



////////////////////////////////////////////////////////////////////////////////////////
// CIRCULO WINDOWS C++ 1.0                                                            //
//                                                                                    //
// Análisis y Diseño:                                                                 //
//                                                                                    //
// Humberto Andres Carrillo Calvet                                                    //
// Antonio Carrillo Ledesma                                                           //
```

```
// Oscar García Regis                                                                    //
// Miguel Angel Mendoza Reyes                                                            //
// Fernando Alberto Ongay Larios                                                         //
//                                                                                       //
// Programación:                                                                         //
//                                                                                       //
// Humberto Andres Carrillo Calvet                                                       //
// Antonio Carrillo Ledesma                                                              //
//                                                                                       //
///////////////////////////////////////////////////////////////////////////////////////////


#include <vcl.h>
#include <stdio.h>
#include <Printers.hpp>
#include <vcl\Registry.hpp>
#include "Acecade.h"
#include "Ayuda.h"
#include "Editor.h"
#include "Sincronizaciones.hpp"
#include "NumeroRotacion.hpp"
#include "ExponenteLyapunov.hpp"
#include "ACX-WebBrowser.h"
#pragma hdrstop

#include "VentanaPrincipal.h"
#include "VentanaBifurcaciones.h"
#include "VentanaCirculo.h"
#include "VentanaToro.h"
#include "VentanaEscalera.h"
#include "VentanaLevantamientos.h"
#include "VentanaLenguas.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
//---------------------------------------------------------------------------

TFormaPrincipal *FormaPrincipal;

// Objetos globales del sistema IyD
Sincronizaciones       sincro;
NumeroRotacion         num_rot;
ExponenteLyapunov      exp_lyap;


// Variables globales de control de escenarios
bool                   Limpiar_ventana             = false;
bool                   Calcular_sincronizaciones   = false;
bool                   Calcular_numero_rotacion    = false;
bool                   Calcular_exponente_lyapunov = false;
bool                   Configura_escenario         = false;
bool                   Calcula_escenario           = false;
bool                   Detener_Calculo             = false;
bool                   Zoom_manteniendo_aspecto    = true;
void                  *Ventana_activa              = NULL;

char *TRAYECTORIA_ARCHIVOS = ".\\";
char *COMPANIA             = "Laboratorio de Sistemas Complejos S.C, y Laboratorio de Dinámica no Lineal,
UNAM";
char *E_MAIL               = "E-mail: dinamica@www.dynamics.unam.edu";
char *WWW                  = "http://www.dynamics.unam.edu/circle";

// Valores Globales para el COPY and PAST
long double PARAMETROS[NUM_MAX_PARAMETROS];



// Constructor de la forma
__fastcall TFormaPrincipal::TFormaPrincipal(TComponent* Owner) : TForm(Owner)
{
   // Titulo de la aplicación
   sprintf(TituloAplicacion,"Circle: %s",def_ecu.Nombre_sistema);
   Application->Title = "Circle 1.0";
   Caption = TituloAplicacion;

   TRegistry &regkey = * new TRegistry();
   bool keygood = regkey.OpenKey("Laboratorio de Dinamica no Lineal\\Circle 1.0",false);
   int top = 30, left = 30;
   if (keygood) {
      top = regkey.ReadInteger("Top");
```

```
      left = regkey.ReadInteger("Left");
   }
   Left = left, Top = top;
   try {
      // Carga el icono de la aplicación
      Application->Icon->LoadFromFile("Circulo.ICO");
   } catch (...) {};
   // Tiempo maximo de muestra de Hits
   Application->HintHidePause = 10000;
   // Presentación
   MenuAyudaAcercadeClick(this);
}

// Al crear la forma ...
void __fastcall TFormaPrincipal::FormCreate(TObject *Sender)
{
   // Asigna la rutina de visualizacion de la barra de estado
   Application->OnHint = &OnHint;
}

// Al activar la forma
void __fastcall TFormaPrincipal::FormActivate(TObject *Sender)
{
   Application->OnHint = &OnHint;
}


// Controla la visualizacion de Hint en la barra de estado
void __fastcall TFormaPrincipal::OnHint(TObject *Sender)
{
   BarraDeEstadoPrincipal->SimpleText = Application->Hint;
}

// Controla la solicitud de cerrar la forma
void __fastcall TFormaPrincipal::FormCloseQuery(TObject *Sender,bool &CanClose)
{
   // Graba la configuracion de la ventana
   TRegistry &regkey = * new TRegistry();
   bool keygood = regkey.OpenKey("Laboratorio de Dinamica no Lineal\\Sawtooth 1.0",true);
   if (keygood) {
      regkey.WriteInteger("Top", Top);
      regkey.WriteInteger("Left", Left);
   }

   if (MessageBox(Handle,"Do you wish to end the program?",TituloAplicacion,MB_YESNO + MB_ICONQUESTION)
== IDYES) CanClose = true;
    else CanClose = false;
}


////////////////////////////////////////////////////////////////////////////
//  Definición de Comportamientos del Menu
////////////////////////////////////////////////////////////////////////////

//  MenuPrincipal->Archivo->Editor de archivos
void __fastcall TFormaPrincipal::MenuArchivoEditFileClick(TObject *Sender)
{
   TEditorForm *editor = new TEditorForm(this);
   if (editor) {
      editor->Abrir_archivo("", false);
   }
}

// MenuPrincipal->Archivo->Imprimir archivo
void __fastcall TFormaPrincipal::MenuArchivoPrintFileClick(TObject *Sender)
{
   OpenPictureDialog->Title = "Load File ...";
   if (OpenPictureDialog->Execute()) {
      Graphics::TBitmap *Bitmap = new Graphics::TBitmap;
      Bitmap->LoadFromFile(OpenPictureDialog->FileName);
      Printer()->BeginDoc();
      Printer()->Canvas->TextOut(100,100,Application->Title);
      Printer()->Canvas->StretchDraw(Rect(100,200,Printer()->PageWidth-100,Printer()->PageWidth-
200),Bitmap);
      Printer()->Canvas->TextOut(100,(Printer()->PageWidth - 100),COMPANIA);
      Printer()->Canvas->TextOut(100,(Printer()->PageWidth - 10),E_MAIL);
      Printer()->EndDoc();
      delete Bitmap;
   }
}
```

```
// MenuPrincipal->Archivo->Configura impresión
void __fastcall TFormaPrincipal::MenuArchivoPrintsetupClick(TObject *Sender)
{
   PrinterSetupDialog->Execute();
}


// MenuPrincipal->Archivo->Salir
void __fastcall TFormaPrincipal::MenuArchivoTerminarProgramaClick(TObject *Sender)
{
   Close();
}

//MenuPrincipal->Configurar->Numero de Rotación
void __fastcall TFormaPrincipal::MenuConfigurarNumRotacionClick(TObject *Sender)
{
   configura = new TVCap_General(this);
   if(configura) {
      PasarValoresConfiguracion();
      configura->TabbedNotebook1->PageIndex = 0;
      configura->ShowModal();
      RetornarValoresConfiguracion();
   }
   delete configura;
}

//MenuPrincipal->Configurar->Sincronización
void __fastcall TFormaPrincipal::MenuConfigurarSincronizacionClick(TObject *Sender)
{
   configura = new TVCap_General(this);
   if(configura) {
      PasarValoresConfiguracion();
      configura->TabbedNotebook1->PageIndex = 1;
      configura->ShowModal();
      RetornarValoresConfiguracion();
   }
   delete configura;
}

//MenuPrincipal->Configurar->Exponente de Lyapunov
void __fastcall TFormaPrincipal::MenuConfigurarExpLyapunovClick(TObject *Sender)
{
   configura = new TVCap_General(this);
   if(configura) {
      PasarValoresConfiguracion();
      configura->TabbedNotebook1->PageIndex = 2;
      configura->ShowModal();
      RetornarValoresConfiguracion();
   }
   delete configura;
}

// Acerca de ...
void __fastcall TFormaPrincipal::MenuAyudaAcercadeClick(TObject *Sender)
{
   TVAcercaDe *Acercade = new TVAcercaDe(this);
   if (Acercade) {
      Acercade->ShowModal();
      delete Acercade;
   }
}

// Ayuda General
void __fastcall TFormaPrincipal::MenuAyudaGeneralClick(TObject *Sender)
{
   TAyudaForm *Ayuda = new TAyudaForm(this);
   if (Ayuda) {
      Ayuda->Abrir_archivo("General help","General.hlp");
      Ayuda->Show();
   }
}

// Ayuda de Bifurcaciones
void __fastcall TFormaPrincipal::MenuAyudaBifurcacionesClick(TObject *Sender)
{
   TAyudaForm *Ayuda = new TAyudaForm(this);
   if (Ayuda) {
      Ayuda->Abrir_archivo("Bifurcations scenery","Bifurcations.hlp");
      Ayuda->Show();
   }
```

```
}

// Ayuda de Circulo
void __fastcall TFormaPrincipal::MenuAyudaCirculoClick(TObject *Sender)
{
   TAyudaForm *Ayuda = new TAyudaForm(this);
   if (Ayuda) {
      Ayuda->Abrir_archivo("Circle scenery","Circle.hlp");
      Ayuda->Show();
   }
}

// Ayuda de Toro
void __fastcall TFormaPrincipal::MenuAyudaToroClick(TObject *Sender)
{
   TAyudaForm *Ayuda = new TAyudaForm(this);
   if (Ayuda) {
      Ayuda->Abrir_archivo("Torus scenery","Torous.hlp");
      Ayuda->Show();
   }
}

// Ayuda de Curvas Integrales
void __fastcall TFormaPrincipal::MenuAyudaCurvasIntegralesClick(TObject *Sender)
{
   TAyudaForm *Ayuda = new TAyudaForm(this);
   if (Ayuda) {
      Ayuda->Abrir_archivo("Sawtooth scenery","IntegralCurves.hlp");
      Ayuda->Show();
   }
}

// Ayuda de Levantamientos
void __fastcall TFormaPrincipal::MenuAyudaLevantamientosClick(TObject *Sender)
{
   TAyudaForm *Ayuda = new TAyudaForm(this);
   if (Ayuda) {
      Ayuda->Abrir_archivo("Lifts scenery","Lifts.hlp");
      Ayuda->Show();
   }
}

// Ayuda de Lenguas
void __fastcall TFormaPrincipal::MenuAyudaLenguasClick(TObject *Sender)
{
   TAyudaForm *Ayuda = new TAyudaForm(this);
   if (Ayuda) {
      Ayuda->Abrir_archivo("Tongues scenery","Tongues.hlp");
      Ayuda->Show();
   }
}

//////////////////////////////////////////////////////////////////////////////
//  Definición de Comportamientos de los Iconos de Escenarios
//////////////////////////////////////////////////////////////////////////////

// Icono de Escenario de Bifurcaciones
void __fastcall TFormaPrincipal::IconoEscenarioBifurcacionesClick(TObject *Sender)
{
   TFormaBifurcaciones *vent_bif = new TFormaBifurcaciones(this);
   if(vent_bif) {
      vent_bif->Show();
   }
}

// Icono de Escenario de Circunferencia
void __fastcall TFormaPrincipal::IconoEscenarioCirculoClick(TObject *Sender)
{
   TFormaCirculo *vent_cir = new TFormaCirculo(this);
   if(vent_cir) {
      vent_cir->Show();
   }
}

// Icono de Escenario de Toro
void __fastcall TFormaPrincipal::IconoEscenarioToroClick(TObject *Sender)
{
   TFormaToro *vent_tor = new TFormaToro(this);
   if(vent_tor) {
      vent_tor->Show();
   }
```

```
}

// Icono de Escalera
void __fastcall TFormaPrincipal::IconoEscenarioEscaleraClick(TObject *Sender)
{
   TFormaEscalera *vent_cur = new TFormaEscalera(this);
   if(vent_cur) {
      vent_cur->Show();
   }
}

// Icono de Escenario de Levantamientos
void __fastcall TFormaPrincipal::IconoEscenarioLevantamientosClick(TObject *Sender)
{
   TFormaLevantamientos *vent_lev = new TFormaLevantamientos(this);
   if(vent_lev) {
      vent_lev->Show();
   }
}

// Icono de Escenario de Lenguas de Arnold
void __fastcall TFormaPrincipal::IconoEscenarioTonguesClick(TObject *Sender)
{
   TFormaLenguas *vent_lev = new TFormaLenguas(this);
   if(vent_lev) {
      vent_lev->Show();
   }
}


//////////////////////////////////////////////////////////////////////////////
//  Definición de Comportamientos de los Iconos Acción
//////////////////////////////////////////////////////////////////////////////

// Icono de calcula el número de rotacion del escenario activo
void __fastcall TFormaPrincipal::IconoAccionNumRotacionClick(TObject *Sender)
{
   if (Ventana_activa) Calcular_numero_rotacion = true;
}

// Icono de calcula la sincronización del escenario activo
void __fastcall TFormaPrincipal::IconoAccionSincronizacionClick(TObject *Sender)
{
   if (Ventana_activa) Calcular_sincronizaciones = true;
}

// Icono de calcula el exponente de Luapunov del escenario activo
void __fastcall TFormaPrincipal::IconoAccionExpLyapunovClick(TObject *Sender)
{
   if (Ventana_activa) Calcular_exponente_lyapunov = true;
}

// Icono de configura el escenario  del escenario activo
void __fastcall TFormaPrincipal::IconoAccionConfigurarEscenarioClick(TObject *Sender)
{
   if (Ventana_activa) Configura_escenario = true;
}

// Icono de calcula del escenario activo
void __fastcall TFormaPrincipal::IconoAccionCalcularClick(TObject *Sender)
{
   if (Ventana_activa) Calcula_escenario = true;
}

// Icono de detener cálculo del escenario activo
void __fastcall TFormaPrincipal::IconoAccionDetenerCalculoClick(TObject *Sender)
{
   if (Ventana_activa) Detener_Calculo = true;
}

// Icono de limpiar ventana del escenario activo
void __fastcall TFormaPrincipal::IconoAccionLimpiarVentanaClick(TObject *Sender)
{
   if (Ventana_activa) Limpiar_ventana = true;
}


//////////////////////////////////////////////////////////////////////////////
// Control de la ventana de captura de parámetros globales
//////////////////////////////////////////////////////////////////////////////
```

```
// Pasa los valores de configuración de la ventana de captura
void TFormaPrincipal::PasarValoresConfiguracion(void)
{
   ////////////////////////////////////////////////////
   // Numero de rotación
   ////////////////////////////////////////////////////
   sprintf(xcad,"%u",num_rot.Rot_Num_iteraciones);
   configura->EditR1->Text = (AnsiString) xcad;
   sprintf(xcad,"%Lf",num_rot.Rot_Condicion_Inicial);
   configura->EditR2->Text = (AnsiString) xcad;
   configura->CheckBoxR1->Checked = num_rot.Rot_sw_Cond_inicial_aleatoria;
   ////////////////////////////////////////////////////
   // Sincronizaciones
   ////////////////////////////////////////////////////
   sprintf(xcad,"%u",sincro.Res_Long_Min_transitorios);
   configura->EditS1->Text = (AnsiString) xcad;
   sprintf(xcad,"%u",sincro.Res_Long_Max_transitorios);
   configura->EditS2->Text = (AnsiString) xcad;
   sprintf(xcad,"%u",sincro.Res_Long_max_ciclo);
   configura->EditS3->Text = (AnsiString) xcad;
   sprintf(xcad,"%1.6Le",sincro.Res_Tolerancia);
   configura->EditS4->Text = (AnsiString) xcad;
   sprintf(xcad,"%u",sincro.Num_condiciones_iniciales);
   configura->EditS5->Text = (AnsiString) xcad;
   sprintf(xcad,"%1.6Le",sincro.Res_Condicion_Inicial);
   configura->EditS6->Text = (AnsiString) xcad;
   configura->CheckBoxS1->Checked = sincro.Res_sw_Cond_inicial_aleatoria;
   ////////////////////////////////////////////////////
   // Exponente de Lyapunov
   ////////////////////////////////////////////////////
   sprintf(xcad,"%u",exp_lyap.Max_num_iteraciones);
   configura->EditL2->Text = (AnsiString) xcad;
   sprintf(xcad,"%Lf",exp_lyap.Epsilon);
   configura->EditL3->Text = (AnsiString) xcad;
   sprintf(xcad,"%Lf",exp_lyap.Paso_derivada);
   configura->EditL4->Text = (AnsiString) xcad;
   sprintf(xcad,"%Lf",exp_lyap.Condicion_inicial);
   configura->EditL5->Text = (AnsiString) xcad;
   configura->CheckBoxL1->Checked = exp_lyap.Sw_cond_inicial_aleatoria;
}

// Retorna los valores de configuración de la ventana de captura
void TFormaPrincipal::RetornarValoresConfiguracion(void)
{
   if (!configura->Aceptar) return;
   // Numero de rotación
   ////////////////////////////////////////////////////
   num_rot.Rot_Num_iteraciones   = atoi(configura->EditR1->Text.c_str());
   num_rot.Rot_Condicion_Inicial = _atold(configura->EditR1->Text.c_str());
   num_rot.Rot_sw_Cond_inicial_aleatoria = configura->CheckBoxR1->Checked;
   ////////////////////////////////////////////////////
   // Sincronizaciones
   ////////////////////////////////////////////////////
   sincro.Res_Long_Min_transitorios     = atoi(configura->EditS1->Text.c_str());
   sincro.Res_Long_Max_transitorios     = atoi(configura->EditS2->Text.c_str());
   sincro.Res_Long_max_ciclo            = atoi(configura->EditS3->Text.c_str());
   sincro.Res_Tolerancia                = _atold(configura->EditS4->Text.c_str());
   sincro.Num_condiciones_iniciales     = atoi(configura->EditS5->Text.c_str());
   sincro.Res_Condicion_Inicial         = _atold(configura->EditS5->Text.c_str());
   sincro.Res_sw_Cond_inicial_aleatoria = configura->CheckBoxS1->Checked;
   ////////////////////////////////////////////////////
   // Exponente de Lyapunov
   ////////////////////////////////////////////////////
   exp_lyap.Max_num_iteraciones  = atoi(configura->EditL2->Text.c_str());
   exp_lyap.Epsilon              = _atold(configura->EditL3->Text.c_str());
   exp_lyap.Paso_derivada        = _atold(configura->EditL4->Text.c_str());
   exp_lyap.Condicion_inicial    = _atold(configura->EditL5->Text.c_str());
   exp_lyap.Sw_cond_inicial_aleatoria = configura->CheckBoxL1->Checked;
}



void __fastcall TFormaPrincipal::MenuVentanaConformalZoomClick(TObject *Sender)
{
   if (Zoom_manteniendo_aspecto) {
      Zoom_manteniendo_aspecto = false;
      MenuVentanaConformalZoom->Caption = "Quasiconformal &Zoom";
   } else {
      Zoom_manteniendo_aspecto = true;
      MenuVentanaConformalZoom->Caption = "Conformal &Zoom";
   }
```

```
}

void __fastcall TFormaPrincipal::LabDinmicanoLineal1Click(TObject *Sender)
{
   TFormaWebBrowzer *vent = new TFormaWebBrowzer(this);
   if (vent) {
       vent->Parametros("http://www.dynamics.unam.edu");
       vent->Show();
   }
}


////////////////////////////////////////////////////////////////////////////////////////////
// CIRCULO WINDOWS C++ 1.0                                                                //
//                                                                                        //
// Análisis y Diseño:                                                                     //
//                                                                                        //
// Humberto Andres Carrillo Calvet                                                        //
// Antonio Carrillo Ledesma                                                               //
// Oscar García Regis                                                                     //
// Miguel Angel Mendoza Reyes                                                             //
// Fernando Alberto Ongay Larios                                                          //
//                                                                                        //
// Programación:                                                                          //
//                                                                                        //
// Humberto Andres Carrillo Calvet                                                        //
// Antonio Carrillo Ledesma                                                               //
//                                                                                        //
////////////////////////////////////////////////////////////////////////////////////////////



#include "Sincronizaciones.hpp"
#include "Man_arch.hpp"

// Retorna el numero de sincronizacion
int Sincronizaciones::Calcula(const bool  t_i)
{
   bool Found = false;
   // Almacena el primer valor de las iteraciones para buscar las resonancias
   long double x, Frac_x, Int_x = 0.0, incX;
   // Almacena las iteraciones de la funcion de disparos
   long double New_x, FracNew_x, IntNew_x;
   // Auxiliar en las busquedas
   long double Ancla_x, FracAncla_x, IntAncla_x;
   //Det. la cantidad de sincros para un conj. de parametros especificos
   unsigned char Indice_sincronizaciones = 0;
   // Definicion del arreglo de sincronizaciones
   long double IncInits;
   unsigned int Posible_Q, Posible_P;
   unsigned int i;
   unsigned int Encontro;  //Determina si se ha encontrado alguna posible orbita periodica
   char xcad[1000];
   long double CondInicial;

   Manipulador_archivos *Gp = new Manipulador_archivos;
   Gp->Parametros("C:\\REPTMP.TMP",GRABA_ARCHIVO,200,"");
   Gp->Graba_linea("Reporte de Sincronizaciones");
   Gp->Graba_linea("");
   Encontro = FALSO;


   // Inicializa arreglo de sincronizaciones
   for (i = 0; i < MAYORSINCRONIZACION; i++)
   {
       Arreglo_Sincronizaciones_Q[i] = 0;
       Arreglo_Sincronizaciones_P[i] = 0;
   }

   //Barrido de Cond. Iniciales
   if (Res_sw_Cond_inicial_aleatoria) incX = 1.0/((long double) Num_condiciones_iniciales), IncInits =
0.0;
    else incX = 0.6, IncInits = Res_Condicion_Inicial;

   for ( ; IncInits < 1.0; IncInits += incX )
   {
      // Condicion inicial condición inicial aleatoria dentro del intervalo
      Frac_x = IncInits + (incX * (random(10)/10.0));
      CondInicial = Frac_x;

      //  A partir de la Condicion Inicial se realiza un Transitorio de iteraciones sobre ella
```

```
for(i = 0; i < Res_Long_Min_transitorios; i++) {
    x = Frac_x;
    x = FuncionDisparo::Calcula(x,t_i) + Int_x;
    Frac_x = modfl(x,&Int_x);
}

Encontro = FALSO;
Posible_Q = 0;
FracNew_x = Frac_x;
```

Diez Ultimas Páginas

```
         // Region de difeomorfismos
         while (b < maxauxY) {
            Application->ProcessMessages();
            if (Sw_cerrar_ventana || Detener_calculo) break;
            sprintf(xcad,"Calculating Tongues of the parameter %s = %2.8Lf (this process may last several
minutes).",fun_disp.Nombre_parametros[Segundo_Eje], b);
            BarraDeEstadosLenguas->SimpleText=(AnsiString) xcad;
            // Selecciona un parametro dentro de la malla
            a = MinX;
            while (a <= MaxX) {
            xFACTOR = 1 + Factor_usado_zona_difeomorfismos * ((((Cota_superior_region_difeomorfismos -
b)*100.0) / Cota_superior_region_difeomorfismos)/100.0);
#ifdef MAYA_CON_ALEATORIOS
               aux = (random(10)/10.0);
               aaux = a + ((Incx/xFACTOR) * aux);
               baux = b + ((Incy/xFACTOR) * aux);
               Calcula_Lenguas_Solicitadas(a, b, aaux, baux, true);
#else
               Calcula_Lenguas_Solicitadas(a, b, a, b, true);
#endif
               a += (Incx/xFACTOR);
            }
            b += (Incy/xFACTOR);
         }
         minauxY = Cota_superior_region_difeomorfismos;
      } else minauxY = MinY;

      // Region de no difeomorfismos
      for (b = minauxY; b <= MaxY; b += Incy ) {
         Application->ProcessMessages();
         if (Sw_cerrar_ventana || Detener_calculo) break;
         sprintf(xcad,"Calculating Tongues of the parameter %s = %2.8Lf, Multistability = %i (this
process may last several minutes).",fun_disp.Nombre_parametros[Segundo_Eje],
b,Maxima_estabilidad_encontrada_al_calculo_lenguas);
         BarraDeEstadosLenguas->SimpleText=(AnsiString) xcad;
         // Selecciona un parametro dentro de la malla
         for (a = MinX; a <= MaxX; a += Incx) {
#ifdef MAYA_CON_ALEATORIOS
            aux = (random(10)/10.0);
            aaux = a + (Incx * aux);
            baux = b + (Incy * aux);
            Calcula_Lenguas_Solicitadas(a, b, aaux, baux, false);
#else
            Calcula_Lenguas_Solicitadas(a, b, a, b, false);
#endif
         }
      }
    } else {
      if (MinY < Cota_superior_region_difeomorfismos) {
         if (MaxY > Cota_superior_region_difeomorfismos) maxauxY = Cota_superior_region_difeomorfismos;
          else maxauxY = MaxY;
         b =  MinY;
         // Region de difeomorfismos
         while (b < maxauxY) {
            Application->ProcessMessages();
            if (Sw_cerrar_ventana || Detener_calculo) break;
            sprintf(xcad,"Calculating Tongues of the parameter %s = %2.8Lf (this process may last several
minutes).",fun_disp.Nombre_parametros[Segundo_Eje], b);
            BarraDeEstadosLenguas->SimpleText=(AnsiString) xcad;
            // Selecciona un parametro dentro de la malla
            a = MinX;
            while (a <= MaxX) {
               xFACTOR = 1 + Factor_usado_zona_difeomorfismos * ((((Cota_superior_region_difeomorfismos -
b)*100.0) / Cota_superior_region_difeomorfismos)/100.0);
#ifdef MAYA_CON_ALEATORIOS
               aux = (random(10)/10.0);
               aaux = a + ((Incx/xFACTOR) * aux);
               baux = b + ((Incy/xFACTOR) * aux);
               Calcula_Lenguas_Todas(a, b, aaux, baux, true);
#else
               Calcula_Lenguas_Todas(a, b, a, b, true);
#endif
               a += (Incx/(xFACTOR));
            }
            b += (Incy/xFACTOR);
         }
         minauxY = Cota_superior_region_difeomorfismos;
      } else minauxY = MinY;

      // Region de no difeomorfismos
      for (b = minauxY; b <= MaxY; b += Incy ) {
```

```
         Application->ProcessMessages();
         if (Sw_cerrar_ventana || Detener_calculo) break;
         sprintf(xcad,"Calculating Tongues of the parameter %s = %2.8Lf, Multistabolity = %i (this
process may last several minutes).",fun_disp.Nombre_parametros[Segundo_Eje],
b,Maxima_estabilidad_encontrada_al_calculo_lenguas);
         BarraDeEstadosLenguas->SimpleText=(AnsiString) xcad;
         // Selecciona un parametro dentro de la malla
         for (a = MinX; a <= MaxX; a += Incx) {
#ifdef MAYA_CON_ALEATORIOS
            aux = (random(10)/10.0);
            aaux = a + (Incx * aux);
            baux = b + (Incy * aux);
            Calcula_Lenguas_Todas(a, b, aaux, baux, false);
#else
            Calcula_Lenguas_Todas(a, b, a, b, false);
#endif
         }
      }
   }
   // Cerrar archivo
   fwrite(&Terminador, sizeof(Terminador), 1, OutField);
   fclose(OutField);
   Grafica(false);
   time(&tf);


   char xcad1[300];
   TLMensajeForm *Ayuda = new TLMensajeForm(this);
   if(!Ayuda) return;
   if(Tipo_lenguas != 1) Ayuda->Caption = "Sincronizations found";
   else Ayuda->Caption = "Sincronizations find";
   Ayuda->Editor->Lines->Add("");


   unsigned int Count = 0;
   if (N_Sincro_encontradas) {
      Ordenacion_lenguas(Tongues_Q, Tongues_P);
      while (Count < N_Sincro_encontradas) {
         sprintf(xcad1,"   %u:%u", Tongues_P[Count], Tongues_Q[Count]);
         Ayuda->Editor->Lines->Add(xcad1);
         Count ++;
      }
   }
   if(Tipo_lenguas != 1) sprintf(xcad1,"   %u sincronizations were founded", N_Sincro_encontradas);
   else sprintf(xcad1,"   %u sincronizations were founded", N_Sincro_encontradas);
   Ayuda->Editor->Lines->Add("");
   Ayuda->Editor->Lines->Add(xcad1);
   sprintf(xcad1,"Maximum miltistability founded %i", Maxima_estabilidad_encontrada_al_calculo_lenguas);
   Ayuda->Editor->Lines->Add("");
   Ayuda->Editor->Lines->Add(xcad1);
   Ayuda->Editor->Lines->Add("");
   strcpy(xcad1,ctime(&ti));
   xcad1[24] = 0;
   Ayuda->Editor->Lines->Add(xcad1);
   strcpy(xcad1,ctime(&tf));
   xcad1[24] = 0;
   Ayuda->Editor->Lines->Add(xcad1);
   Ayuda->Show();
   delete[] Tongues_Q;
   delete[] Tongues_P;
   delete[] Sincronizaciones_en_un_punto_Q;
   delete[] Sincronizaciones_en_un_punto_P;
   delete[] Arreglo_Sincronizaciones_Q;
   delete[] Arreglo_Sincronizaciones_P;

   Tongues_P = NULL;
   Tongues_Q = NULL;

   BarraDeEstadosLenguas->SimpleText=(AnsiString) " ";
   Cursor = antcursor;
   Sw_Proceso_calculo = false;
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////
//////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////
// Genera lista de sincronizaciones
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

// Genera la lista de sincronizaciones
void TFormaLenguas::Genera_lista_sincronizaciones(void)
{
   // Inicialización
   int i;
   N_Sincro_encontradas = 0;
   for (i = 0; i < MAX_ARREGLO; i++) {
      Tongues_P[i] = 0;
      Tongues_Q[i] = 0;
   }
   // Creación de la lista de sincronizaciones
   Leer_Especificaciones_Calculo();
   // Encuentra la mayor orbita
   SizeOrbits = 0;
   for (i = 0; i < N_Sincro_encontradas; i++) {
      if (Tongues_Q[i] > SizeOrbits) SizeOrbits= Tongues_Q[i];
   }
}


////////////////////////////////////////////////////////////////////////////////////////////////////
/////////
////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

void __fastcall TFormaLenguas::Copyparameters1Click(TObject *Sender)
{
   long double parm[NUM_MAX_PARAMETROS];
   fun_disp.RetornaParametros(parm);
   parm[Primer_Eje] = Vs.Xi;
   parm[Segundo_Eje] = Vs.Yi;
   fun_disp.AsignaParametros(parm);
}
//---------------------------------------------------------------------------

void __fastcall TFormaLenguas::ShowSincronization1Click(TObject *Sender)
{
   long double parm[NUM_MAX_PARAMETROS];
   fun_disp.RetornaParametros(parm);
   parm[Primer_Eje] = Vs.Xi;
   parm[Segundo_Eje] = Vs.Yi;
   fun_disp.AsignaParametros(parm);
   MenuCalcularSincronizacinClick(this);
}


void __fastcall TFormaLenguas::ShowRotational1Click(TObject *Sender)
{
   long double parm[NUM_MAX_PARAMETROS];
   fun_disp.RetornaParametros(parm);
   parm[Primer_Eje] = Vs.Xi;
   parm[Segundo_Eje] = Vs.Yi;
   fun_disp.AsignaParametros(parm);
   MenuCalcularNumeroRotacionClick(this);
}



// Lee las especificaciones del calculo
int TFormaLenguas::Leer_Especificaciones_Calculo(void)
{
   unsigned int *Back_T_P, *Back_T_Q; //Lenguas y Color Auxiliares
   unsigned int Resonancy[6];

   FILE *ApTongue;      //Apuntador al archivo de datos de sincronizaciones
   ApTongue = fopen("C:\\CONFIG.DAT","rt");
   if (!ApTongue) return 1;

   //Construccion de los arreglos para los Qs, Ps
   Back_T_Q = new unsigned int[MAX_ARREGLO];
   Back_T_P = new unsigned int[MAX_ARREGLO];

   Intern = 0;
   SizeTgs = 0;
   while (!feof(ApTongue)) {
      //Obtiene Informacion de las Sincronizaciones
      Leer_Linea_Archivo(Resonancy,ApTongue);
      Analizar_Entrada(Resonancy, Back_T_Q, Back_T_P);
```

```
   } //Termina while(!THE_END)
   fclose(ApTongue);  //Cierra el archivo de informacion de Sincronizaciones
   // Ordenacion de las lenguas, considerando el periodo
   Ordenacion(Back_T_P, Back_T_Q);
   N_Sincro_encontradas = Intern;

   delete []Back_T_Q;
   delete []Back_T_P;
   return 0;
}

//Analiza las entradas de la lectura del archivo de especificacion de Sincronizaciones
int TFormaLenguas::Analizar_Entrada(const unsigned int  *Resonancy, unsigned int *Back_T_Q, unsigned int
*Back_T_P)
{
   unsigned int Cuenta;
   unsigned int i, xi;
   bool st;

   //Verifica que la informacion de las sincronizaciones sea correcta

   //Caso si existe sincronizacion o rango de sinc.
   if( (Resonancy[0]!=0) || (Resonancy[1]!=0) )
   {
       // caso cuando existe un rango de sincronizaciones
       if( (Resonancy[2]!=0) || (Resonancy[3]!=0) )
       {
           //Caso: Qs Fijos, Ps Varian
           if( Resonancy[0] == Resonancy[2] )
           {
               Cuenta = abs( Resonancy[3] - Resonancy[1]) + 1;
               for (i = 0; i < Cuenta; i++)
               {
                   st = false;
                   for (xi = 0; xi < Intern; xi++) {
                       if ( Back_T_Q[xi] == Resonancy[0] && Back_T_P[xi] == (Resonancy[1] + i)) st =
true;
                   }
                   if (!st) {
                      Back_T_Q[Intern] = Resonancy[0];
                      Back_T_P[Intern] = Resonancy[1] + i;
                      Intern++;   //Aumento en el indice de los arreglos
                      SizeTgs++;
                   }
               }
           }
           else
           {
              //Caso: Ps Fijos, Qs Varian
              if( Resonancy[1] == Resonancy[3] )
              {
                   Cuenta = abs( Resonancy[2] - Resonancy[0]) + 1;
                   for (i = 0; i < Cuenta; i++)
                   {
                       st = false;
                       for (xi = 0; xi < Intern; xi++) {
                           if ( Back_T_Q[xi] == (Resonancy[0] + i) && Back_T_P[xi] == Resonancy[1]) st =
true;
                       }
                       if (!st) {
                          Back_T_Q[Intern]= Resonancy[0] + i;
                          Back_T_P[Intern]= Resonancy[1];

                          SizeTgs++;
                          Intern++;   //Aumento en el indice de los arreglos
                       }
                   }
              }
           }
       }
       else
       {
          for (xi = 0; xi < Intern; xi++) {
              if ( Back_T_Q[xi] == Resonancy[0] && Back_T_P[xi] == Resonancy[1]) return 0;
          }
          Back_T_Q[Intern] = Resonancy[0];
          Back_T_P[Intern] = Resonancy[1];

          Intern++;   //Aumento en el indice de los arreglos
          SizeTgs++;
       }
```

```
   }
   //Caso en que no existe sincronizacion o rango de sinc. 0:0
   else return 1;
   return 0;
}


// Funcion de lectura para el archivo de lenguas.
void TFormaLenguas::Leer_Linea_Archivo(unsigned int  *Resonancy, FILE *ApTongue)
{
   char entra[100], car;
   int i, END_LINE = DESACTIVADO, Indice = 0;
   THE_END = DESACTIVADO;

   //Inicializacion del arreglo de rango de sincronizaciones
   for (i = 0; i < 4; i++) Resonancy[i] = 0;
   i = 0;
   while ( !feof(ApTongue) && (!END_LINE) ) {
      car = fgetc(ApTongue);
      if (feof(ApTongue)) {
         if (i) {
            entra[i] = 0;
            Resonancy[Indice++] = atoi(entra);
         }
         return;
      }
      if (car == ' ') continue;
      if (car == '\n') END_LINE = ACTIVADO;
      if ((car != ':') && (car != ',') && (car != '\n')) entra[i++] = car;
      else {
         entra[i] = 0;
         if ( strcmp(entra, "...") ) {
            if (!THE_END) Resonancy[Indice++] = atoi(entra);
            //Limpieza del buffer entra
            entra[0] = 0;
            i = 0;
           } else {
            entra[0] = 0;
            i = 0;
          }
      }
   }
}



// Función del calculo de Sincronizaciones Solicitadas   <<SINCROS == ACTIVADO>>
void TFormaLenguas::Calcula_Lenguas_Solicitadas(const long double a, const long double b, const long
double aaux, const long double baux, const bool difeomorfismo)
{
   // Ajusta los parametros
   fun_disp.RetornaParametros(parm1);
   fun_disp.RetornaParametros(parm1);
   parm1[Primer_Eje] = aaux;
   parm1[Segundo_Eje] = baux;
   fun_disp.AsignaParametros(parm1);


   // Almacena el primer valor de las iteraciones para buscar las resonancias
   long double x, Frac_x, Int_x = 0.0, incX;
   // Almacena las iteraciones de la funcion de disparos
   long double New_x, FracNew_x, IntNew_x;
   // Auxiliar en las busquedas
   long double Ancla_x, FracAncla_x, IntAncla_x;
   //Det. la cantidad de sincros para un conj. de parametros especificos
   unsigned char Indice_sincronizaciones = 0;
   long double IncInits;
   unsigned int Posible_Q, Posible_P;
   unsigned int i;
   unsigned int Encontro;  //Determina si se ha encontrado alguna posible orbita periodica
   int Count;
   bool existen_sincronizaciones_en_calculo_anterior;


   // Inicializa arreglo de sincronizaciones
   for (i = 0; i < Maxima_estabilidad_buscada_al_calculo_lenguas; i++) {
       Arreglo_Sincronizaciones_Q[i] = 0;
       Arreglo_Sincronizaciones_P[i] = 0;
   }
```

```
   // Si es necesario inicializa el arreglo de las últimas sincronizaciones calculadas
   if (Sincronizaciones_en_un_punto_Q[0] == 0 && Sincronizaciones_en_un_punto_P[0] == 0) {
      existen_sincronizaciones_en_calculo_anterior = false;
      for (i = 0; i < Maxima_estabilidad_buscada_al_calculo_lenguas; i++) {
         Sincronizaciones_en_un_punto_Q[i] = 0;
         Sincronizaciones_en_un_punto_P[i] = 0;
      }
   } else existen_sincronizaciones_en_calculo_anterior = true;



   // Barrido de Condiciones iniciales
   for (int nci = 0; nci < NCondInits; nci++) {
      // No continua el barrido de condiciones iniciales si ya encontro la máxima multiestabilidad dada
      if (Indice_sincronizaciones >= Maxima_estabilidad_buscada_al_calculo_lenguas) break;
      if (difeomorfismo && Indice_sincronizaciones >= 1) break;

      // Condición inicial aleatoria dentro del intervalo
      Frac_x = Condiciones_Iniciales[nci] + ((Condiciones_Iniciales[nci+1]-Condiciones_Iniciales[nci]) *
(random(10)/10.0));

      //  A partir de la Condicion Inicial se realiza un Transitorio de iteraciones sobre ella
      Int_x = 0.0;
      for(i = 0; i < ItsTrans; i++) {
          x = Frac_x;
          x = fun_disp.Calcula(x,Tipo_integracion) + Int_x;
          Frac_x = modfl(x,&Int_x);
      }

      Encontro = FALSO;
      Posible_Q = 0;
      FracNew_x = Frac_x;
      Int_x = 0.0;
      while( (Encontro == FALSO) && (Posible_Q <= SizeOrbits)) {
         x = Frac_x;
         x = fun_disp.Calcula(x,Tipo_integracion) + Int_x;
         Frac_x = modfl(x, &Int_x);
         //Almacena la posible longitud de la órbita periódica
         Posible_Q++;
         i ++;
         //Condicion para det. si hay órbita
         if ( (fabsl(Frac_x - FracNew_x) < Tolerancia_maxima) || (fabsl(fabsl(Frac_x - FracNew_x)-1.0) <
Tolerancia_maxima) ) Encontro = VERDADERO;
      }
      if (Encontro != VERDADERO) continue;


      Encontro = FALSO;
      while(i <= Num_max_trans && (Encontro == FALSO) ) {
         Posible_Q = 0;
         FracNew_x = Frac_x;
         Int_x = 0.0;
         while( (Encontro == FALSO) && (Posible_Q <= SizeOrbits)) {
            x = Frac_x;
            x = fun_disp.Calcula(x,Tipo_integracion) + Int_x;
            Frac_x = modfl(x, &Int_x);
            //Almacena la posible longitud de la órbita periódica
            Posible_Q++;
            i ++;
            //Condicion para det. si hay órbita
            if ( (fabsl(Frac_x - FracNew_x) < Tolerancia) || (fabsl(fabsl(Frac_x - FracNew_x)-1.0) <
Tolerancia) ) Encontro = VERDADERO;
         }
      }
      if (i > Num_max_trans) continue;


      //Inicia comprobacion de la resonancia hallada
      if ((Encontro == VERDADERO)) {
          Encontro = FALSO;
          if ( (Posible_Q > 0) && (Posible_Q <= SizeOrbits) ) {
               //New_x=New_x+(Tolerancia/10); //Perturbacion ligera si se desea
               New_x = Ancla_x = Frac_x;
               //Ciclo de check (long. de Posible_Q) desde 1 hasta Posible_Q, pues ahora solo se recorrá
la orbita que se presume haber encontrado
               for ( i = 1; i <= Posible_Q; i++) {
                    FracNew_x = modfl(New_x,&IntNew_x);
                    New_x = fun_disp.Calcula(FracNew_x,Tipo_integracion) + IntNew_x;
               }

               FracAncla_x = modfl(Ancla_x, &IntAncla_x);
```

```
              FracNew_x = modfl(New_x, &IntNew_x);
              //Condicion para det. si hay órbita
              if ( (fabsl(FracAncla_x - FracNew_x) < Tolerancia) || (fabsl(fabsl(FracAncla_x - FracNew_x)
- 1.0) < Tolerancia) ) Encontro = VERDADERO;

              // Se encontro resonancia y paso el check
              if (Encontro == VERDADERO)  {
                  // Determina la posible envolvencia asociada a la órbita encontrada para "Posible_Q"
este ya se encuentra almacenado
                  Posible_P = IntNew_x - IntAncla_x;

                  // Posible P no debera de excede a Posible Q
                  if (Posible_P < 254)
                  {
                     Count = 0;
                     while (Count < N_Sincro_encontradas) {
                        if ( Tongues_Q[Count] == Posible_Q && Tongues_P[Count] == Posible_P ) {

                           if (Indice_sincronizaciones > 0) {
                              for (i = 0; i < Indice_sincronizaciones; i++) {
                                 if (Arreglo_Sincronizaciones_Q[i] == Posible_Q &&
Arreglo_Sincronizaciones_P[i] == Posible_P) break;
                              }
                              if (i == Indice_sincronizaciones) {
                                 if (i < Maxima_estabilidad_buscada_al_calculo_lenguas) {
                                     Arreglo_Sincronizaciones_Q[Indice_sincronizaciones] = (unsigned
char) Posible_Q;
                                     Arreglo_Sincronizaciones_P[Indice_sincronizaciones] = (unsigned
char) Posible_P;
                                     Indice_sincronizaciones++;
                                 }
                              }
                            } else {
                               Arreglo_Sincronizaciones_Q[Indice_sincronizaciones] = (unsigned char)
Posible_Q;
                               Arreglo_Sincronizaciones_P[Indice_sincronizaciones] = (unsigned char)
Posible_P;
                               Indice_sincronizaciones++;
                           }
                           break;

                        }
                        Count ++;
                     }
                  }
              } // if de "Se encontro resonancia y paso el check"
          } //termina "if( (Posible_Q>0L) && (Posible_Q<=SizeOrbits) )"
      } //termina "if((Encontro==VERDADERO)) //Inicia comprob.(Check) de la resonancia hallada"
   } //Termina for de "IncInit"

   // Almacenamiento de los valores de los parametros y sus sincronizaciones
   if (Indice_sincronizaciones > 0)
   {
      int px, py;
      int Count;
      float xa = (float) a , xb = (float) b;
      fwrite(&xa, sizeof(xa), 1, OutField);
      fwrite(&xb, sizeof(xb), 1, OutField);
      // grabar Indice_sincronizaciones
      fwrite(&Indice_sincronizaciones, sizeof(Indice_sincronizaciones), 1, OutField);

      for (i = 0; i < Indice_sincronizaciones; i++)
      {
         fwrite(&Arreglo_Sincronizaciones_Q[i], sizeof(Arreglo_Sincronizaciones_Q[i]), 1, OutField);
         fwrite(&Arreglo_Sincronizaciones_P[i], sizeof(Arreglo_Sincronizaciones_P[i]), 1, OutField);
         if (!existen_sincronizaciones_en_calculo_anterior) {
            Sincronizaciones_en_un_punto_Q[i] = Arreglo_Sincronizaciones_Q[i];
            Sincronizaciones_en_un_punto_P[i] = Arreglo_Sincronizaciones_P[i];
         }

         Count = 0;
         while (Count < N_Sincro_encontradas) {
            if ( Tongues_Q[Count] == Arreglo_Sincronizaciones_Q[i] && Tongues_P[Count] ==
Arreglo_Sincronizaciones_P[i] ) break;
            Count ++;
         }

         pix[Count%16].Asigna_valor(a,b,true);
         px = (a - Dim_Vtn.Xi) * Escala.X;
         py = VTLenguas->Height - (((b - Dim_Vtn.Yi) * Escala.Y) + 1.0);
```

```
         VTLenguas->Canvas->Pixels[px][py] = Colores[Count%16];
      }
      if (Indice_sincronizaciones > Maxima_estabilidad_encontrada_al_calculo_lenguas)
Maxima_estabilidad_encontrada_al_calculo_lenguas = Indice_sincronizaciones;
   }
}


//  Función del calculo de Todas las Posibles Sincronizaciones  <<SINCROS == DESACTIVADO>>
void TFormaLenguas::Calcula_Lenguas_Todas(const long double a, const long double b, const long double
aaux, const long double baux, const bool difeomorfismo)
{
   // Ajusta los parametros
   fun_disp.RetornaParametros(parm1);
   fun_disp.RetornaParametros(parm1);
   parm1[Primer_Eje] = aaux;
   parm1[Segundo_Eje] = baux;
   fun_disp.AsignaParametros(parm1);


   // Almacena el primer valor de las iteraciones para buscar las resonancias
   long double x, Frac_x, Int_x = 0.0, incX;
   // Almacena las iteraciones de la funcion de disparos
   long double New_x, FracNew_x, IntNew_x;
   // Auxiliar en las busquedas
   long double Ancla_x, FracAncla_x, IntAncla_x;
   //Det. la cantidad de sincros para un conj. de parametros especificos
   unsigned char Indice_sincronizaciones = 0;
   // Definicion del arreglo de sincronizaciones
   long double IncInits;
   unsigned int Posible_Q, Posible_P;
   unsigned int i;
   unsigned int Encontro;  //Determina si se ha encontrado alguna posible orbita periodica
   bool existen_sincronizaciones_en_calculo_anterior;



   // Inicializa arreglo de sincronizaciones
   for (i = 0; i < Maxima_estabilidad_buscada_al_calculo_lenguas; i++) {
       Arreglo_Sincronizaciones_Q[i] = 0;
       Arreglo_Sincronizaciones_P[i] = 0;
   }

   // Si es necesario inicializa el arreglo de las últimas sincronizaciones calculadas
   if (Sincronizaciones_en_un_punto_Q[0] == 0 && Sincronizaciones_en_un_punto_P[0] == 0) {
      existen_sincronizaciones_en_calculo_anterior = false;
      for (i = 0; i < Maxima_estabilidad_buscada_al_calculo_lenguas; i++) {
         Sincronizaciones_en_un_punto_Q[i] = 0;
         Sincronizaciones_en_un_punto_P[i] = 0;
      }
   } else existen_sincronizaciones_en_calculo_anterior = true;


   //Barrido de Condiciones iniciales
   for (int nci = 0; nci < NCondInits; nci++) {
      // No continua el barrido de condiciones iniciales si ya encontro la máxima multiestabilidad dada
      if (Indice_sincronizaciones >= Maxima_estabilidad_buscada_al_calculo_lenguas) break;
      if (difeomorfismo && Indice_sincronizaciones >= 1) break;

      // Condicion inicial condición inicial aleatoria dentro del intervalo
      Frac_x = Condiciones_Iniciales[nci] + ((Condiciones_Iniciales[nci+1]-Condiciones_Iniciales[nci]) *
(random(10)/10.0));

      //  A partir de la Condicion Inicial se realiza un Transitorio de iteraciones sobre ella
      Int_x = 0.0;
      for(i = 0; i < ItsTrans; i++) {
          x = Frac_x;
          x = fun_disp.Calcula(x,Tipo_integracion) + Int_x;
          Frac_x = modfl(x,&Int_x);
      }

      // Se revisa provisionalmente si esta convergiendo a algo
      Encontro = FALSO;
      Posible_Q = 0;
      FracNew_x = Frac_x;
      Int_x = 0.0;
      while( (Encontro == FALSO) && (Posible_Q <= SizeOrbits))
      {
         x = Frac_x;
         x = fun_disp.Calcula(x,Tipo_integracion) + Int_x;
         Frac_x = modfl(x, &Int_x);
```

```
         //Almacena la posible longitud de la órbita periódica
         Posible_Q++;
         i ++;
         //Condicion para det. si hay órbita
         if ( (fabsl(Frac_x - FracNew_x) < Tolerancia_maxima) || (fabsl(fabsl(Frac_x - FracNew_x)-1.0) <
Tolerancia_maxima) ) Encontro = VERDADERO;
      }
      // Si no converge provisionalmente a algo se descarta la condición inicial
      if (Encontro != VERDADERO) continue;

      // Revisa si existe almenos una sincronización encontrada
      if (Indice_sincronizaciones) {
         // Revisa si la candidata es una de las que ya existian
         Posible_P = Int_x;
         for (i = 0; i < Indice_sincronizaciones; i++) {
            if (Arreglo_Sincronizaciones_Q[i] == Posible_Q && Arreglo_Sincronizaciones_P[i] == Posible_P)
{
               // Descarta la candidata por ya existir
               Encontro = FALSO;
               break;
            }
         }
      }
      // Descarta la candidata por ya existir
      if (Encontro != VERDADERO) continue;

      // Hace el refinamiento de la sincronización candidata
      Encontro = FALSO;
      while(i <= Num_max_trans && (Encontro == FALSO) ) {
         Posible_Q = 0;
         FracNew_x = Frac_x;
         Int_x = 0.0;
         while( (Encontro == FALSO) && (Posible_Q <= SizeOrbits))
         {
            x = Frac_x;
            x = fun_disp.Calcula(x,Tipo_integracion) + Int_x;
            Frac_x = modfl(x, &Int_x);
            //Almacena la posible longitud de la órbita periódica
            Posible_Q++;
            i ++;
            //Condicion para det. si hay órbita
            if ( (fabsl(Frac_x - FracNew_x) < Tolerancia) || (fabsl(fabsl(Frac_x - FracNew_x)-1.0) <
Tolerancia) ) Encontro = VERDADERO;
         }
      }
      if (i > Num_max_trans) continue;

      //Inicia comprobacion de la resonancia hallada
// #error falta el chequeo de la sincronizacion encontrada
      if (Encontro == VERDADERO) {
          Encontro = FALSO;
          if ( (Posible_Q > 0) && (Posible_Q <= SizeOrbits) ) {
               //New_x=New_x+(Tolerancia/10); //Perturbacion ligera si se desea
               New_x = Ancla_x = Frac_x;
               //Ciclo de check (long. de Posible_Q) desde 1 hasta Posible_Q, pues ahora solo se recorrá
la orbita que se presume haber encontrado
               for ( i = 1; i <= Posible_Q; i++) {
                    FracNew_x = modfl(New_x,&IntNew_x);
                    New_x = fun_disp.Calcula(FracNew_x,Tipo_integracion) + IntNew_x;
               }

               FracAncla_x = modfl(Ancla_x, &IntAncla_x);

               FracNew_x = modfl(New_x, &IntNew_x);
               //Condicion para det. si hay órbita
               if ( (fabsl(FracAncla_x - FracNew_x) < Tolerancia) || (fabsl(fabsl(FracAncla_x - FracNew_x)
- 1.0) < Tolerancia) ) Encontro = VERDADERO;

               // Se encontro resonancia y paso el check
               if (Encontro == VERDADERO) {
                   // Determina la posible envolvencia asociada a la órbita encontrada para "Posible_Q"
este ya se encuentra almacenado
                   Posible_P = IntNew_x - IntAncla_x;

                   // Posible P no debera de excede a Posible Q
                   if (Posible_P < 254)  {
                      //Caso en que no se proporciona lista de sincronizaciones opciones de Grabado para
Multisincronizacion
                      if (Indice_sincronizaciones > 0) {
                         for (i = 0; i < Indice_sincronizaciones; i++) {
```

```
                           if (Arreglo_Sincronizaciones_Q[i] == Posible_Q &&
Arreglo_Sincronizaciones_P[i] == Posible_P) break;
                        }
                        if (i == Indice_sincronizaciones) {
                           if (i < Maxima_estabilidad_buscada_al_calculo_lenguas) {
                              Arreglo_Sincronizaciones_Q[Indice_sincronizaciones] = (unsigned char)
Posible_Q;
                              Arreglo_Sincronizaciones_P[Indice_sincronizaciones] = (unsigned char)
Posible_P;
                              Indice_sincronizaciones++;
                           }
                        }
                     } else {
                        Arreglo_Sincronizaciones_Q[Indice_sincronizaciones] = (unsigned char) Posible_Q;
                        Arreglo_Sincronizaciones_P[Indice_sincronizaciones] = (unsigned char) Posible_P;
                        Indice_sincronizaciones++;
                     }
                  }
               } // if de "Se encontro resonancia y paso el check"
            } //termina "if( (Posible_Q>0L) && (Posible_Q<=SizeOrbits) )"
         } //termina "if((Encontro==VERDADERO)) //Inicia comprob.(Check) de la resonancia hallada"
      } //Termina for de "IncInit"

      // Almacenamiento de los valores de los parametros y sus sincronizaciones
      if (Indice_sincronizaciones > 0)
      {
         int px, py;
         int Count;
         float xa = (float) a , xb = (float) b;
         fwrite(&xa, sizeof(xa), 1, OutField);
         fwrite(&xb, sizeof(xb), 1, OutField);
         // grabar Indice_sincronizaciones
         fwrite(&Indice_sincronizaciones, sizeof(Indice_sincronizaciones), 1, OutField);
         for (i = 0; i < Indice_sincronizaciones; i++) {
            // Graba la sincronización encontrada
            fwrite(&Arreglo_Sincronizaciones_Q[i], sizeof(Arreglo_Sincronizaciones_Q[i]), 1, OutField);
            fwrite(&Arreglo_Sincronizaciones_P[i], sizeof(Arreglo_Sincronizaciones_P[i]), 1, OutField);
            if (!existen_sincronizaciones_en_calculo_anterior) {
               Sincronizaciones_en_un_punto_Q[i] = Arreglo_Sincronizaciones_Q[i];
               Sincronizaciones_en_un_punto_P[i] = Arreglo_Sincronizaciones_P[i];
            }

            // Revisa si ya esta en el arreglo de lenguas en caso contrario la anexa
            Count = 0;
            while (Count < N_Sincro_encontradas) {
               if ( Tongues_Q[Count] == Arreglo_Sincronizaciones_Q[i] && Tongues_P[Count] ==
Arreglo_Sincronizaciones_P[i] ) break;
               Count ++;
            }
            if ( Count >= N_Sincro_encontradas && N_Sincro_encontradas < MAX_ARREGLO) {
                Tongues_Q[N_Sincro_encontradas] = Arreglo_Sincronizaciones_Q[i];
                Tongues_P[N_Sincro_encontradas] = Arreglo_Sincronizaciones_P[i];
                Count = N_Sincro_encontradas;
                N_Sincro_encontradas ++;
            }
            // Visualiza la sincronizacion encontrada
            pix[Count%16].Asigna_valor(a,b,true);
            px = (a - Dim_Vtn.Xi) * Escala.X;
            py = VTLenguas->Height - (((b - Dim_Vtn.Yi) * Escala.Y) + 1.0);
            VTLenguas->Canvas->Pixels[px][py] = Colores[Count%16];
         }
         if (Indice_sincronizaciones > Maxima_estabilidad_encontrada_al_calculo_lenguas)
Maxima_estabilidad_encontrada_al_calculo_lenguas = Indice_sincronizaciones;
      }
}



// Lanza una ventana del escenario Toro
void __fastcall TFormaLenguas::Torous1Click(TObject *Sender)
{
   TFormaToro *vent_tor = new TFormaToro(this);
   if(vent_tor) {
      for( int i = 0; i < fun_disp.Numero_parametros; i++) vent_tor->fun_disp.P[i] = fun_disp.P[i];
      vent_tor->fun_disp.P[Primer_Eje] = Vs.Xi;
      vent_tor->fun_disp.P[Segundo_Eje] = Vs.Yi;
      vent_tor->Show();
   }
}

// Lanza una ventana del escenario Círculo
```

```
void __fastcall TFormaLenguas::Circle1Click(TObject *Sender)
{
   TFormaCirculo *vent_cir = new TFormaCirculo(this);
   if(vent_cir) {
      for( int i = 0; i < fun_disp.Numero_parametros; i++) vent_cir->fun_disp.P[i] = fun_disp.P[i];
      vent_cir->fun_disp.P[Primer_Eje] = Vs.Xi;
      vent_cir->fun_disp.P[Segundo_Eje] = Vs.Yi;
      vent_cir->Show();
   }
}
```

Listado de Archivos fuentes

04/12/2002 19:33 3.071 Acecade.cpp
04/12/2002 19:31 83.178 Acecade.dfm
04/03/2003 11:25 3.116 Acecade.h
11/07/2002 18:47 1.591 ACX-WebBrowser.cpp
11/07/2002 19:21 606 ACX-WebBrowser.dfm
04/03/2003 11:34 2.751 ACX-WebBrowser.h
15/02/2001 21:10 4.988 Ayuda.cpp
15/04/2002 12:53 1.627 Ayuda.dfm
04/03/2003 11:34 3.558 Ayuda.h
04/03/2003 11:34 34.726 Cadenas.cpp
04/03/2003 11:34 14.798 Cadenas.hpp
01/03/2003 14:13 8.148 Circulo.bpr
04/03/2003 11:32 3.810 Circulo.cpp
27/01/1999 03:00 766 Circulo.ico
04/03/2003 11:34 3.825 Ctrl_l_a.cpp
04/03/2003 11:33 3.010 ctrl_l_a.hpp
04/03/2003 11:25 4.390 DefinicionEcuacion.hpp
04/03/2003 11:35 4.093 Definici¢nEcuaci¢n.cpp
08/01/2003 15:41 18.490 Editor.cpp
15/04/2002 11:24 12.471 Editor.dfm
04/03/2003 11:33 8.117 Editor.h
22/01/2003 16:59 2.484 EnviarCorreo.cpp
22/01/2003 17:06 1.261 EnviarCorreo.dfm
04/03/2003 11:33 4.366 EnviarCorreo.h
04/03/2003 11:35 3.510 ExponenteLyapunov.cpp
04/03/2003 11:25 2.989 ExponenteLyapunov.hpp
04/03/2003 11:33 29.935 Fechas.cpp
04/03/2003 11:33 18.013 Fechas.hpp
04/03/2003 11:33 2.827 Fechora.cpp
04/03/2003 11:33 2.516 Fechora.hpp
04/03/2003 11:23 17.027 FuncionDisparo.cpp
04/03/2003 11:25 5.135 FuncionDisparo.hpp
04/03/2003 11:22 2.204 Lmensaje.cpp
15/04/2002 11:26 621 Lmensaje.dfm
04/03/2003 11:22 2.394 Lmensaje.h
04/03/2003 11:22 19.310 Man_arch.cpp
04/03/2003 11:33 6.964 Man_arch.hpp
04/03/2003 11:22 3.306 NumeroRotacion.cpp
04/03/2003 11:25 3.227 NumeroRotacion.hpp
04/03/2003 11:22 5.281 ParamBifurcaciones.cpp
13/02/2003 10:49 5.047 ParamBifurcaciones.dfm
04/03/2003 11:22 4.292 ParamBifurcaciones.h

```
04/03/2003  11:22             5.284 ParamCirculo.cpp
13/02/2003  11:00             3.178 ParamCirculo.dfm
04/03/2003  11:22             3.692 ParamCirculo.h
04/03/2003  11:22             4.184 ParamEscalera.cpp
13/02/2003  10:48             2.761 ParamEscalera.dfm
04/03/2003  11:21             3.381 ParamEscalera.h
04/03/2003  11:21             2.930 ParamGlobales.cpp
25/10/2001  18:27             3.096 ParamGlobales.dfm
04/03/2003  11:21             3.596 ParamGlobales.h
04/03/2003  11:21             4.236 ParamLenguas.cpp
13/02/2003  11:09             3.566 ParamLenguas.dfm
04/03/2003  11:21             3.690 ParamLenguas.h
04/03/2003  11:20             5.245 ParamLevantamientos.cpp
13/02/2003  10:47             4.277 ParamLevantamientos.dfm
04/03/2003  11:20             4.095 ParamLevantamientos.h
04/03/2003  11:20             5.034 ParamToro.cpp
13/02/2003  10:45             3.284 ParamToro.dfm
04/03/2003  11:20             3.788 ParamToro.h
04/03/2003  11:20             8.984 Sincronizaciones.cpp
04/03/2003  11:24             3.732 Sincronizaciones.hpp
04/03/2003  11:20            10.445 Tiempo.cpp
04/03/2003  11:32             6.335 Tiempo.hpp
04/03/2003  11:19             3.133 VEditor.cpp
14/09/2001  10:22               598 VEditor.dfm
04/03/2003  11:19             2.722 VEditor.h
04/03/2003  11:19            36.623 VentanaBifurcaciones.cpp
13/02/2003  11:06             5.304 VentanaBifurcaciones.dfm
04/03/2003  11:24             9.709 VentanaBifurcaciones.h
04/03/2003  11:19            25.750 VentanaCirculo.cpp
13/02/2003  11:05             4.272 VentanaCirculo.dfm
04/03/2003  11:24             8.556 VentanaCirculo.h
04/03/2003  11:19            30.572 VentanaEscalera.cpp
12/02/2003  11:09             5.220 VentanaEscalera.dfm
04/03/2003  11:24             9.718 VentanaEscalera.h
04/03/2003  11:19            88.299 VentanaLenguas.cpp
12/02/2003  10:26             5.794 VentanaLenguas.dfm
04/03/2003  11:24            13.263 VentanaLenguas.h
04/03/2003  11:19            24.878 VentanaLevantamientos.cpp
13/02/2003  10:43             4.390 VentanaLevantamientos.dfm
04/03/2003  11:24             8.080 VentanaLevantamientos.h
04/03/2003  11:17            19.518 VentanaPrincipal.cpp
12/02/2003  12:48            11.718 VentanaPrincipal.dfm
04/03/2003  11:18             8.067 VentanaPrincipal.h
04/03/2003  11:18            26.682 VentanaToro.cpp
13/02/2003  10:51             4.559 VentanaToro.dfm
04/03/2003  11:24             8.014 VentanaToro.h
04/03/2003  11:20             7.158 V_pixel.cpp
04/03/2003  11:32             9.385 V_pixel.hpp
```